UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE
DEPARTAMENTO DE ENGENHARIA DE COMPUTAÇÃO E AUTOMAÇÃO
ENGENHARIA DE SOFTWARE

GESTCON

ITHAELLY. M. S. MEDEIROS
JOÃO P. J. OLIVEIRA
LUCAS H. A. QUEIROZ
RUAN L. B. NEVES
VINICIUS H. SANTOS

NATAL/RN

2024

RESUMO

O projeto é um sistema de gerenciamento de áreas comuns para condomínios, desenvolvido como um site responsivo e intuitivo. Seu objetivo é simplificar e organizar a utilização dos espaços compartilhados pelos moradores, promovendo praticidade na gestão e eficiência administrativa. Através do sistema, os usuários podem listar os condomínios cadastrados, visualizar e gerenciar as áreas comuns disponíveis, realizar reservas de espaços, além de cadastrar novos usuários, áreas e condomínios. A solução foi projetada para atender às necessidades dos condomínios de forma moderna, acessível e eficiente.

Palavras-chave: Condomínio, site, reserva, áreas comuns.

1. INTRODUÇÃO

Os administradores de condomínios têm a importante responsabilidade de gerenciar e manter um prédio ou conjunto habitacional em pleno funcionamento. No entanto, a tarefa de Administração de Condomínios pode se tornar bastante desafiadora devido a diversos fatores (Conecta, 2024).

Pensando nisso, este projeto foi desenvolvido para oferecer uma solução digital eficiente, moderna e acessível, capaz de otimizar esses processos e melhorar a experiência tanto para os moradores quanto para os administradores.

O projeto é um sistema web desenvolvido com o propósito de facilitar o gerenciamento das áreas compartilhadas de condomínios. A plataforma permite listar e gerenciar condomínios e áreas comuns, realizar reservas de espaços de forma prática, além de possibilitar o cadastro de novos usuários e condomínios. Com isso, busca-se reduzir a complexidade administrativa e oferecer maior conveniência aos moradores.

Quanto às tecnologias empregadas, o backend e a lógica de negócio foram implementados em Java (Oracle, 2024), utilizando o framework Spring Boot, reconhecido pela facilidade na criação de APIs REST. Por sua vez, o frontend foi desenvolvido em JavaScript com a biblioteca React, que proporciona alta interatividade e desempenho em aplicações web.

O banco de dados incorporado foi o PostgreSQL, garantindo um armazenamento de dados confiável e seguro. Além disso, o sistema foi containerizado com Docker, permitindo uma implantação padronizada e simplificada em diferentes ambientes (Susnjara; Smalley, 2024). Durante o desenvolvimento, a IDE IntelliJ IDEA foi utilizada para facilitar uma codificação mais produtiva e organizada.

2. HISTÓRIAS DE USUÁRIO

2.1. Primeira história de usuário

- **Título:** Como administrador, quero cadastrar condomínios no sistema.
- **Descrição:** O administrador precisa ter a possibilidade de registrar os condomínios no sistema, fornecendo informações como nome,

endereço, bloco e descrição. Essa funcionalidade é essencial para estruturar a base de dados do sistema.

- **Prioridade:** Alta.
- **Critérios de Aceitação:**
    - O sistema deve permitir o cadastro de condomínios com os seguintes campos obrigatórios: nome e endereço.
    - O sistema deve validar os dados fornecidos, garantindo que nenhum campo obrigatório esteja vazio.
    - Após o cadastro, o condomínio deve ser listado na interface de gerenciamento do administrador.

2.2. Segunda história de usuário

- **Título:** Como administrador do condomínio, quero cadastrar espaços de convivência para que moradores façam reservas.
- **Descrição:** O administrador deve poder registrar os espaços comuns disponíveis no condomínio, como salão de festas, piscina, ou academia, para que possam ser reservados pelos moradores.
- **Prioridade:** Alta.
- **Critérios de Aceitação:**
    - O sistema deve permitir o cadastro de espaços com campos como nome, descrição e capacidade.
    - Os espaços cadastrados devem ser visíveis na lista de áreas comuns disponíveis para reserva.
    - Deve ser possível excluir um espaço já cadastrado.

2.3. Terceira história de usuário

- **Título:** Como administrador e morador, quero verificar quais espaços estão disponíveis no condomínio.
- **Descrição:** Administradores e moradores devem poder consultar a disponibilidade das áreas comuns do condomínio para planejar reservas ou monitorar o uso dos espaços.
- **Prioridade:** Alta.
- **Critérios de Aceitação:**

    - O sistema deve apresentar uma lista dos espaços comuns, com detalhes sobre datas já reservadas e disponíveis.
    - Para cada espaço, o sistema deve exibir informações detalhadas como capacidade e descrição.

2.4. Quarta história de usuário

- **Título:** Como morador, quero poder reservar um espaço no condomínio.
- **Descrição:** Os moradores precisam ter a possibilidade de realizar reservas de áreas comuns, escolhendo o espaço e a data desejada, desde que estejam disponíveis.
- **Prioridade:** Alta.
- **Critérios de Aceitação:**
    - O sistema deve permitir ao morador selecionar um espaço e uma data disponível para realizar a reserva.
    - O sistema deve validar os dados e impedir a reserva em datas já ocupadas.

2.5. Quinta história de usuário

- **Título:** Como administrador do condomínio, quero criar uma conta para acessar o sistema.
- **Descrição:** O administrador precisa ter um método seguro para registrar sua conta no sistema, com informações como nome, e-mail, senha e associação ao condomínio que gerencia.
- **Prioridade:** Alta.
- **Critérios de Aceitação:**
    - O sistema deve permitir o cadastro de administradores com campos obrigatórios como nome, e-mail, senha e condomínio associado.
    - O administrador deve poder acessar sua conta após o cadastro, utilizando um sistema de autenticação com senha.

3. DIAGRAMA UML

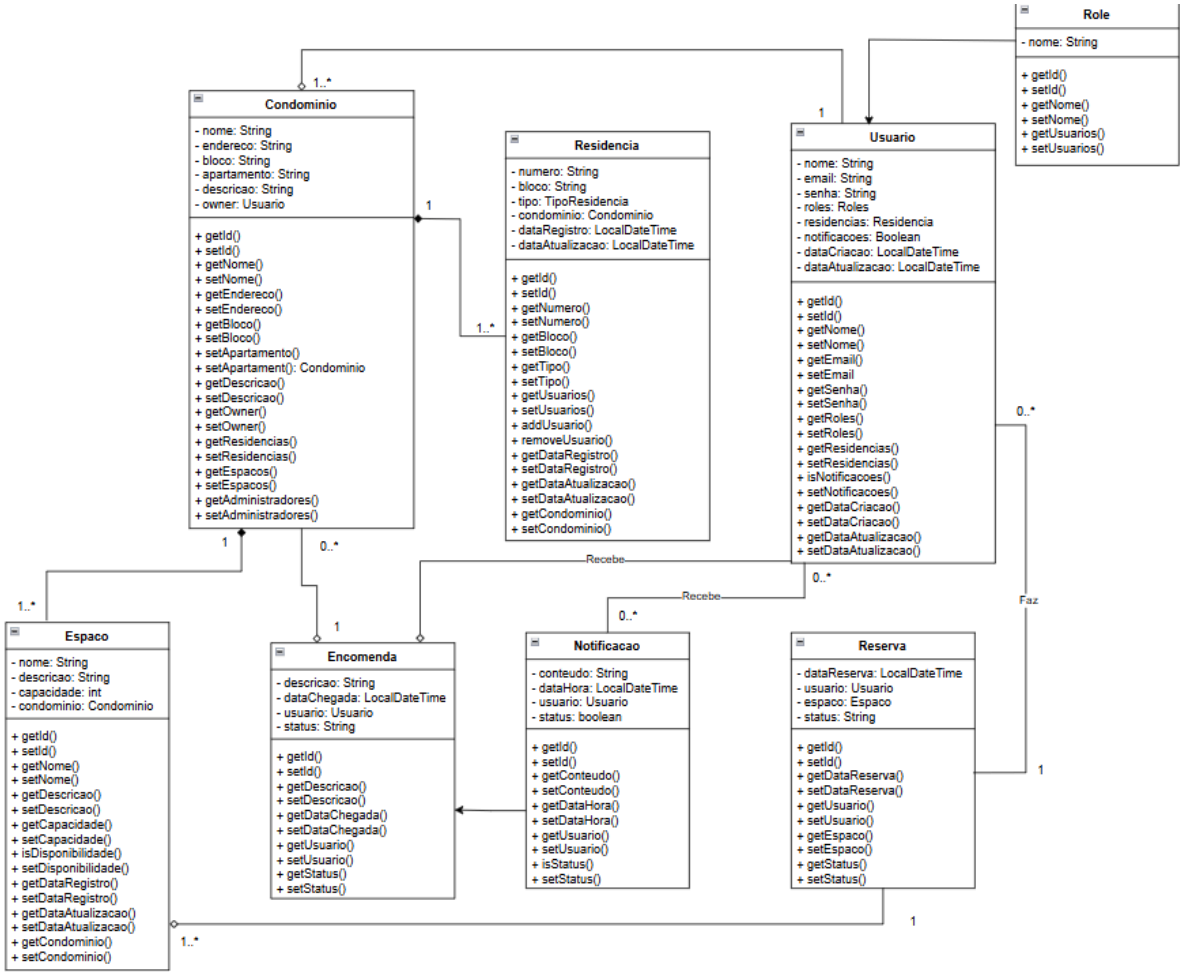
O diagrama de classes UML é uma ferramenta que ajuda a compreender a estrutura e o funcionamento do software. Ele apresenta os componentes do sistema, detalhando suas finalidades individuais e os relacionamentos entre eles.

No caso do sistema de gestão de condomínios, o modelo é composto por oito classes principais: Condomínio, Residência, Usuário, Role, Espaço, Encomenda, Notificação e Reserva.

- A classe *Role* é uma especialização (herança) da classe Usuario e define o tipo de usuário no sistema, podendo ser morador ou administrador.
- A classe *Usuario* possui relacionamentos com outras classes:
    - *Reserva* e *Notificacao*, indicando que um usuário pode realizar reservas em espaços do condomínio e receber notificações.
    - Uma agregação com as classes *Condominio* e *Encomenda*, pois o usuário faz parte de um condomínio, pode receber encomendas e ser notificado quando estas forem registradas pela recepção do condomínio.
    - A classe *Notificacao* é uma especialização da classe Encomenda, representando notificações específicas relacionadas às encomendas capturadas. Além disso, o usuário pode consultar o status de pendências, como encomendas ainda não retiradas.
- A classe *Espaco* representa os espaços disponíveis no condomínio (como salão de festas ou academia) e está associada à classe *Condominio*. Através dessa relação, é possível consultar os espaços disponíveis para reservas.
- A classe *Residencia* também está associada à classe *Condominio* como uma composição, indicando que tanto residências quanto espaços só existem no contexto de um condomínio.

A Figura 1 mostra o diagrama de classes UML do sistema, destacando as relações de herança, associação, agregação e composição entre as classes.

Figura 1 - Diagrama de Classes UML



Fonte: Autores.

4. IMPLEMENTAÇÃO DA API

A API foi projetada para atender às principais funcionalidades necessárias para o gerenciamento de condomínios, incluindo operações de criação, leitura, atualização e exclusão (CRUD) em entidades fundamentais, como condomínios, espaços, usuários e reservas.

A arquitetura da API segue o modelo MVC (Model-View-Controller), garantindo uma separação clara entre as responsabilidades de manipulação de dados, lógica de negócio e interação com o cliente, o que facilita a manutenção e evolução do sistema. Para a persistência, utilizou-se o banco de dados relacional PostgreSQL, acessado através do Spring Data JPA, facilitando o gerenciamento das operações com dados.

4.1. Funcionalidades Implementadas

- **Cadastro de Condomínios:**
  - **Endpoint:** POST /api/condominios
  - **Descrição:** Permite cadastrar novos condomínios. Fornecendo informações como nome, endereço, descrição do condomínio, além do bloco e do apartamento do proprietário.
- **Listar Condomínios:**
  - **Endpoint:** GET /api/condominios
  - **Descrição:** Permite listar todos os condomínios cadastrados no sistema, retornando uma lista com nome, endereço e proprietário.
- **Deletar Condomínios:**
  - **Endpoint:** DELETE /api/condominios/{id}
  - **Descrição:** Deleta um condomínio específico com base no ID fornecido.
- **Atualizar um condomínio:**
  - **Endpoint:** PUT /api/condominios/{id}/{idUser}
  - **Descrição:** Atualiza as informações de um condomínio específico com base no ID do condomínio fornecido. E verificando se o ID do usuário que tentou fazer a atualização é o proprietário ou um dos administradores.
- **Listagem de Espaços comuns do condomínio:**
  - **Endpoint:** GET /api/condominios/{condominioId}/espacos
  - **Descrição:** Retorna uma lista de todas as áreas comuns cadastradas no sistema. Fornecendo o nome, a capacidade e a disponibilidade.

Vale ressaltar que, além do CRUD para condomínios, o sistema também implementa funcionalidades similares para o gerenciamento de espaços comuns e usuários. Isso garante que todas as operações essenciais, como cadastro, listagem, atualização e exclusão, sejam realizadas de forma eficiente e segura para essas entidades, proporcionando uma gestão completa e integrada dentro da plataforma.

5. AUTENTICAÇÃO COM JWT

A autenticação da API foi implementada utilizando JWT (JSON Web Tokens), um mecanismo robusto e seguro para autenticar usuários e controlar o acesso aos recursos da plataforma. O JWT assegura que apenas usuários autenticados possam realizar operações no sistema, e permite a identificação e autorização de diferentes tipos de usuários, de acordo com o papel (role) definido para cada um.

Para autenticar um usuário na API, são necessárias as seguintes informações:

- **E-mail:** O endereço de e-mail do usuário.
- **Senha:** A senha associada ao e-mail do usuário.

Essas informações são enviadas para o servidor por meio de uma requisição POST. Se as credenciais estiverem corretas, o servidor gera um token JWT que será utilizado nas requisições subsequentes para autenticar o usuário e garantir que ele tenha permissões para acessar determinados endpoints.

Claims para os tipos de usuários:

- **Administrador:** Usuário com permissões totais. Pode criar, editar, deletar condomínios, gerenciar usuários e espaços comuns. Esse usuário tem acesso a todos os recursos do sistema.
- **Usuário:** Usuário comum, com permissões limitadas. Pode realizar operações como visualizar espaços comuns, fazer reservas, mas não tem permissão para alterar dados do sistema, como editar ou excluir condomínios.

Comandos necessários para autenticar:

- Enviar uma requisição POST com o e-mail e senha para o endpoint /api/login.
- O servidor verifica as credenciais e, se estiverem corretas, retorna um token JWT.

Exemplo de requisição:

```
POST /api/login
Content-Type: application/json
{
  "email": "usuario@exemplo.com",
  "senha": "senha123"
```

}

6. IMPLANTAÇÃO COM DOCKER

A implantação do sistema desenvolvido é facilitada por meio do uso do Docker, que permite a conteinerização da aplicação e a padronização da execução em diferentes ambientes. A seguir, serão apresentados os passos necessários para a construção e execução do sistema utilizando Docker.

6.1. Dockerfile

O Dockerfile é responsável por definir o ambiente e as instruções para criar a imagem Docker do sistema. Abaixo está o conteúdo do Dockerfile para a aplicação, considerando que o backend é desenvolvido em Java com Spring Boot e o frontend em React.

Dockerfile para Backend (Spring Boot com Java)

Figura 2 - Código do dockerfile no backend

```
1  FROM eclipse-temurin:17.0.8.1_1-jdk-jammy
2  WORKDIR /app
3  COPY . .
4  RUN chmod +x ./mvnw
5  RUN ./mvnw clean install -DskipTests
6  ENTRYPOINT [ "java", "-jar", "target/demo-0.0.1-SNAPSHOT.jar" ]
```

Dockerfile para Frontend (React com Node.js)

Figura 3 - Código do dockerfile no frontend

```
1  FROM node:18
2  WORKDIR /app
3  COPY package*.json ./
4  RUN npm install
5  COPY . .
6  RUN npm run build
7  CMD ["npx", "serve", "-s", "build", "-l", "9090"]
```

6.2. Docker-compose.yml

O docker-compose.yml é utilizado para definir e rodar multi-containers Docker. Neste caso, o arquivo pode ser usado para orquestrar a execução tanto do backend quanto do frontend, além do banco de dados PostgreSQL.

Figura 4 - Código do docker-compose.yml

```yaml
services:
  postgres-db:
    image: postgres:15
    ports:
      - "5432:5432"
    environment:
      - POSTGRES_DB=condominiodb
      - POSTGRES_USER=postgres
      - POSTGRES_PASSWORD=postgres

  spring-app:
    build:
      context: .
      dockerfile: Dockerfile
    image: backend-condominio:latest
    ports:
      - "8080:8080"
    environment:
      SPRING_DATASOURCE_URL: jdbc:postgresql://postgres-db:5432/condominiodb
    depends_on:
      - postgres-db

  spa-app:
    build:
      context: ./demo-web
      dockerfile: Dockerfile
    ports:
      - "9090:9090"
    depends_on:
      - spring-app
```

Comando para executar os arquivos e gerar as imagens do Docker:

docker-compose up --build

7. CONFIGURAÇÃO DO AMBIENTE

7.1. Backend (Spring Boot com Java)

No IntelliJ IDEA:

- Abra o projeto no IntelliJ.
- Importe dependências (Maven/Gradle).
- Configure o backend para execução em Debug:
- Vá em Run > Edit Configurations e selecione a classe principal do Spring Boot.
- Execute em Debug (Shift + F9).

7.2. Frontend (React)

- Navegue até a pasta demo-web no terminal.
    - cd /caminho/para/o/projeto/demo-web
- Instale as dependências:
    - npm install
- Inicie o frontend no modo de desenvolvimento:
    - npm start dev

8. RESULTADOS

Figura 5 - Tela de Bem vindo do sistema

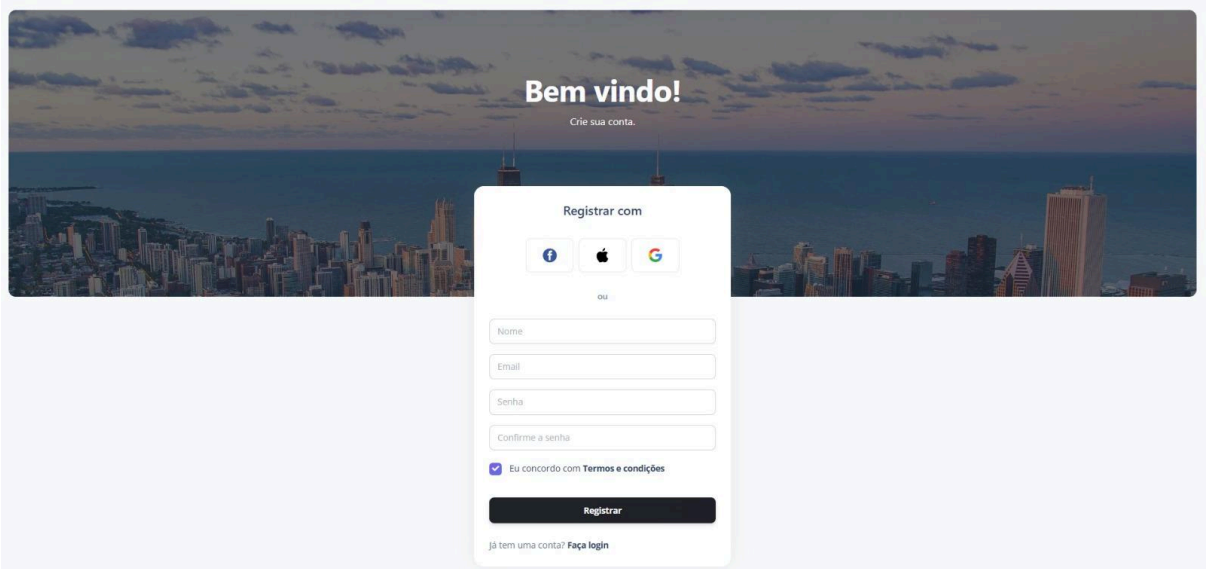


Figura 6- Tela de Login

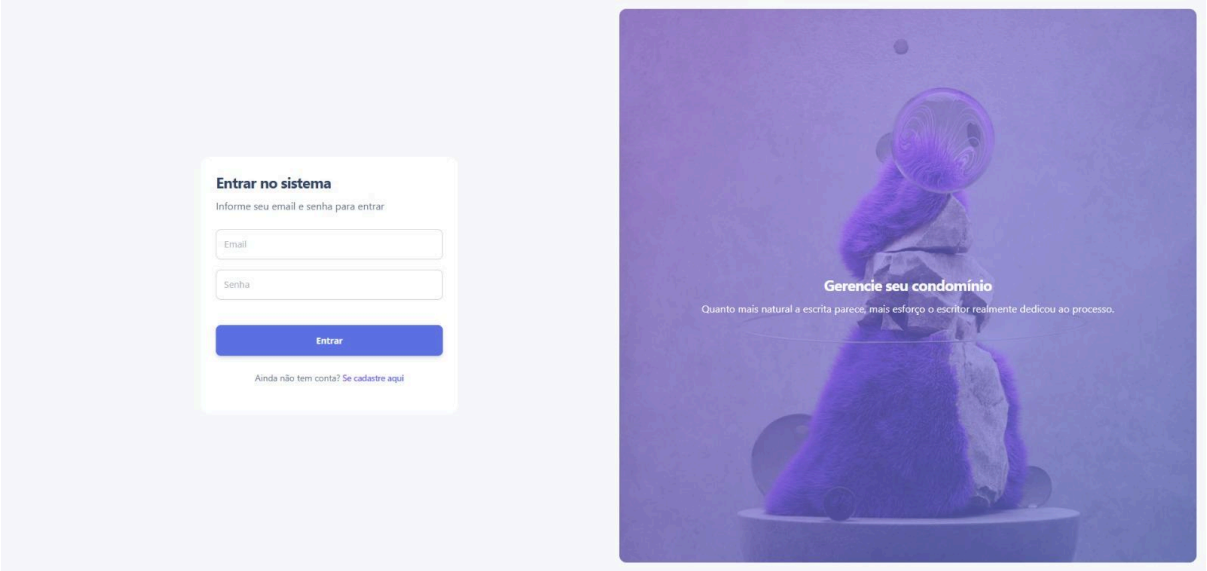


Figura 7- Tela de Cadastro e listagem de Usuários

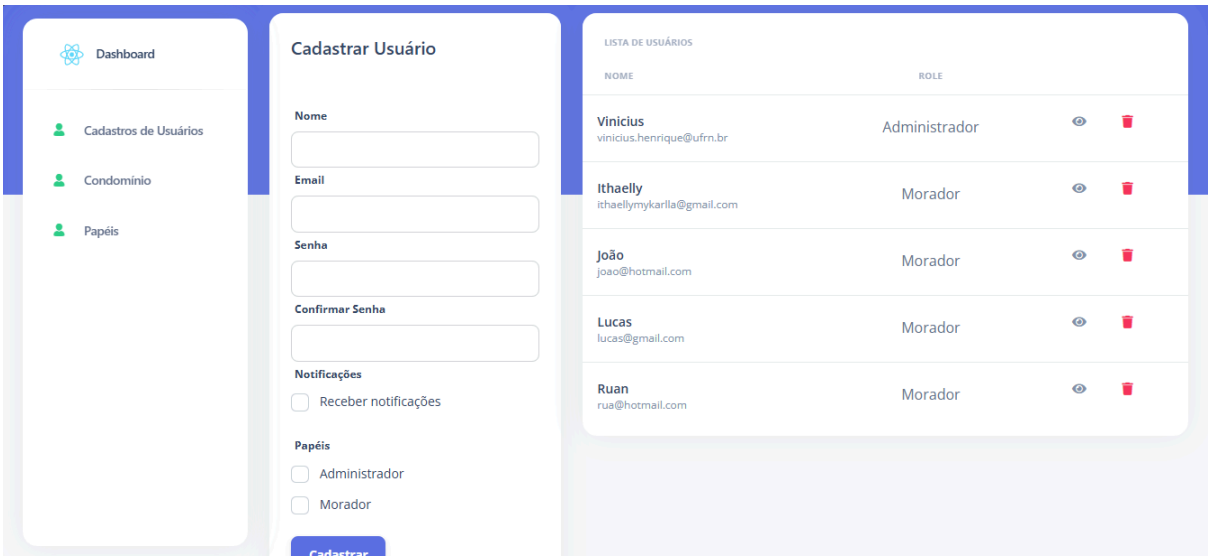


Figura 8- Tela de Cadastro e listagem de Condomínios

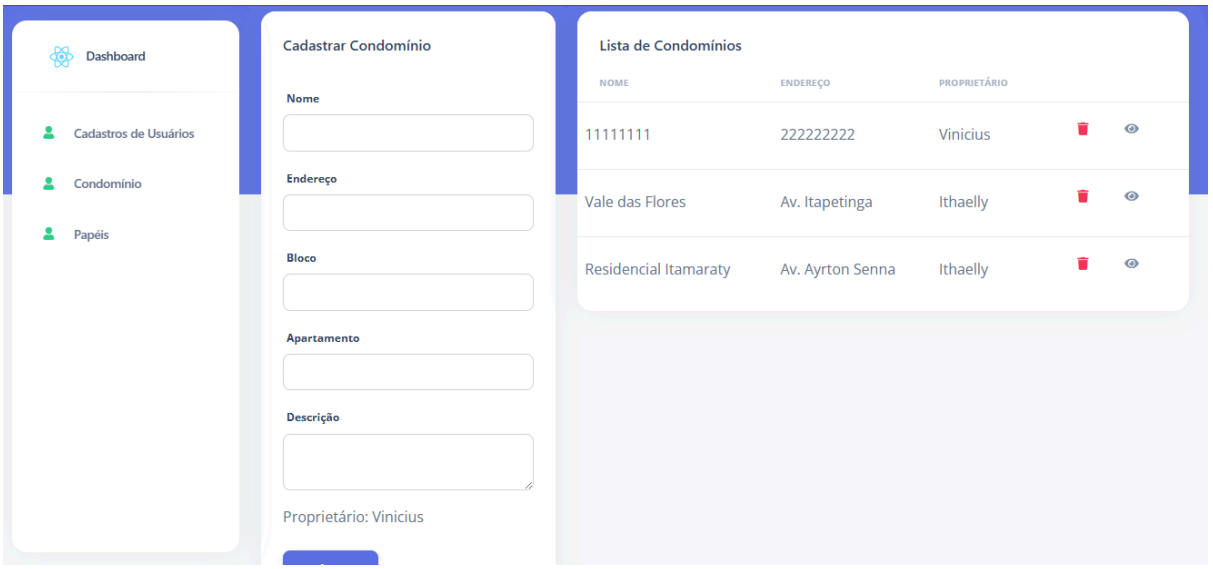


Figura 9- Tela de Detalhes do Condomínio

Figura 10- Tela de Cadastro e listagem de Espaços do condomínio

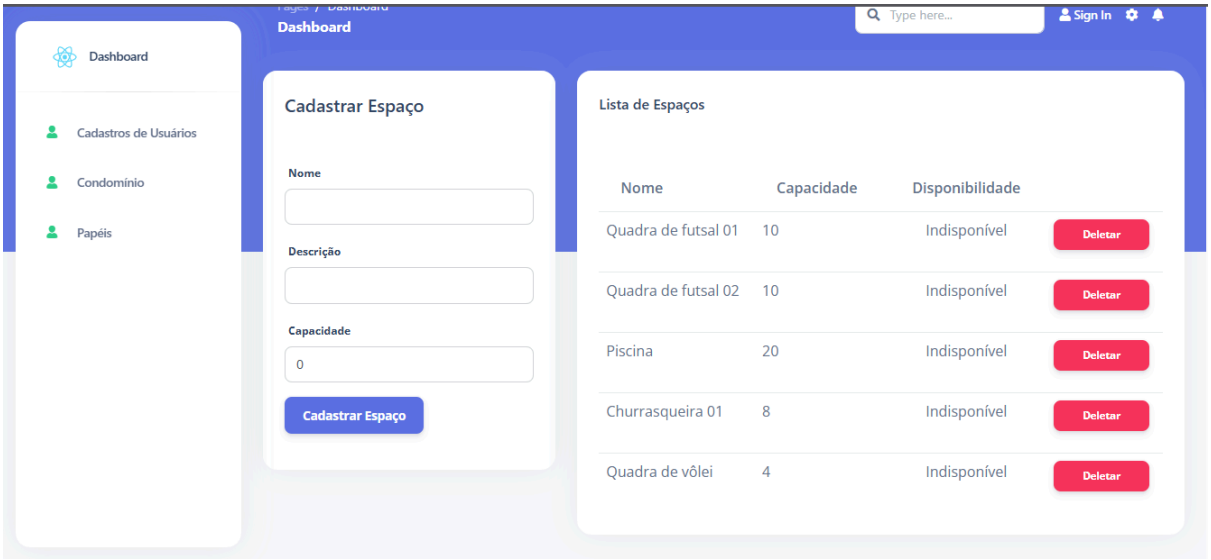


Figura 11- Tela de Agendamentos dos espaços do condomínio

9. CONCLUSÃO

O sistema de gerenciamento de áreas compartilhadas para condomínios foi desenvolvido como uma solução prática e eficiente para atender às necessidades de administradores e moradores, simplificando processos e melhorando a experiência dos usuários.

Além disso, o projeto foi concebido como uma oportunidade para colocar em prática os conceitos e conteúdos abordados na disciplina de Engenharia de

Software. Desde a definição dos requisitos até a implementação das funcionalidades, foram aplicadas técnicas de modelagem, design de sistemas e boas práticas de desenvolvimento, reforçando a conexão entre teoria e prática e consolidando o aprendizado de forma significativa.

10. REFERÊNCIAS

CONECTA, Equipe. **Os 5 maiores desafios dos Administradores de Condomínios**. Disponível em: https://www.conectaadm.com.br/blog/administracao-de-condominios/os-5-maiores-desafios-enfrentados-na-administracao-de-condominios/. Acesso em: 8 dez. 2024.

ORACLE. **O que é a tecnologia Java e por que preciso dela?**. Disponível em: https://www.java.com/en/download/help/whatis_java.html. Acesso em: 8 dez. 2024.

SUSNJARA, Stephanie; SMALLEY, Ian. **O que é Docker?**. Disponível em: https://www.ibm.com/br-pt/topics/docker. Acesso em: 8 dez. 2024.